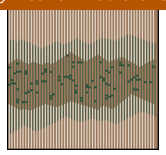
The Journal of Systems and Software 86 (2013) 2225–2241

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



Cloud engineering is Search Based Software Engineering too

Mark Harman[a], Kiran Lakhotia[a], Jeremy Singer[b], David R. White[b,*], Shin Yoo[a]

[a] University College London, Gower Street, London WC1E 6BT, United Kingdom
[b] School of Computing Science, University of Glasgow, G12 8QQ, United Kingdom

ARTICLE INFO

Article history:
Received 2 July 2012
Accepted 15 October 2012
Available online 23 November 2012

Keywords:
Search Based Software Engineering (SBSE)
Cloud computing

ABSTRACT

Many of the problems posed by the migration of computation to cloud platforms can be formulated and solved using techniques associated with Search Based Software Engineering (SBSE). Much of cloud software engineering involves problems of optimisation: performance, allocation, assignment and the dynamic balancing of resources to achieve pragmatic trade-offs between many competing technical and business objectives. SBSE is concerned with the application of computational search and optimisation to solve precisely these kinds of software engineering challenges. Interest in both cloud computing and SBSE has grown rapidly in the past five years, yet there has been little work on SBSE as a means of addressing cloud computing challenges. Like many computationally demanding activities, SBSE has the potential to benefit from the cloud; 'SBSE in the cloud'. However, this paper focuses, instead, of the ways in which SBSE can benefit cloud computing. It thus develops the theme of 'SBSE *for* the cloud', formulating cloud computing challenges in ways that can be addressed using SBSE.

1. Introduction

In many ways cloud computing is not conceptually new: it effectively integrates a set of existing technologies and research areas on a grand scale, but there is nothing inherently new about most of the specific technical challenges involved. The business model that underpins cloud computing *is* new. Rather than viewing software and the hardware upon which it executes as fixed assets to be purchased and subsequently subject to depreciation, the cloud model treats both software and hardware as rented commodities.

This is a profound change, because it removes both the business issue of depreciation and more technically oriented concerns over resource fragmentation and smooth scalability, along with definite commitments to particular platforms, protocols, and interoperability. Nevertheless, in order to avail themselves of these advantages, both the cloud provider and their clients will need new ways of designing, developing, deploying and evolving software, thereby posing new questions for the research community.

The pace of the recent industrial uptake of cloud computing, with its associated technical challenges, also increases the significance of existing conceptual and pragmatic questions that had previously failed to receive widespread attention. Examples include the tailoring of operating system environments and testing in a virtualised environment. Some of these technical issues present new difficulties that must be overcome, whilst others offer opportunities that will enable software engineers to test and deploy code in new ways.

Cloud computing represents a further step along a road that has taken our conception of computation from the rarefied and ideal world of formal calculi to the more mundane and inexact world of engineering reality. In the 1970s and 1980s, there was a serious debate about whether such a thing as 'software engineering' even existed (Hoare, 1978). Many argued that software was not an engineering artefact and that to treat it as such was not only wrong but dangerous, with special opprobrium (Dijkstra, 1978) being reserved for any dissenters who dared to advocate software testing (De Millo et al., 1979).

However, as time passed and systems became larger, there was a gradual realisation that as size and complexity increases, algorithms give way to systems and programs give way to software. A detailed and complete formal proof of a 50 million line banking system may, indeed, be conceptually desirable, but it is no more practical than a quantum mechanical description of the operation of the human circulatory system. The view of software as an engineering artefact, whereby it is meaningful to speak of software engineering as a discipline, gradually became accepted – even by its erstwhile detractors (Hoare, 1996, 1996).

As the recognition of software development as an engineering activity gained increasing traction, the view of software engineering as a discipline fundamentally concerned with optimisation also gained wider attention and acceptance. This optimisation-centric view of software engineering is typified by the development of the

* Corresponding author.
E-mail addresses: m.harman@cs.ucl.ac.uk (M. Harman), k.lakhotia@cs.ucl.ac.uk (K. Lakhotia), jeremy.singer@glasgow.ac.uk (J. Singer), david.r.white@glasgow.ac.uk (D.R. White), shin.yoo@ucl.ac.uk (S. Yoo).

0164-1212 © 2012 Elsevier Inc. Open access under CC BY license.
http://dx.doi.org/10.1016/j.jss.2012.10.027

area of research and practice known as Search Based Software Engineering (SBSE) (Harman and Jones, 2001; Harman, 2007), a topic of growing interest for more than ten years (Freitas and Souza, 2011). SBSE has now permeated almost every area of software engineering activity (Zhang et al., 2008; Afzal et al., 2009; Ali et al., 2010; Räihä, 2010; Afzal and Torkar, 2011; Harman et al., 2012a).

Cloud engineering propels us further along this journey towards a vision of software engineering in which optimisation lies at the very centre. The central justification for deploying software on a cloud platform rests upon a claim about optimisation. That is:

> *Optimisation of resource usage can be achieved by consolidating hardware and software infrastructure into massive datacentres, from which these resources are rented by consumers on-demand.*

Cloud computing is thus the archetypal example of an optimisation-centric view of software engineering. Many of cloud computing's problems and challenges revolve around optimisation, while most of its claimed advantages are unequivocally and unashamedly phrased in terms of optimisation objectives. SBSE is thus a natural fit for cloud computing; it is primarily concerned with the formulation of software engineering problems as optimisation problems, which can subsequently be solved using search algorithms.

As well as providing a natural set of intellectual tools with which to address the challenges and opportunities of cloud computing, SBSE may also provide a convenient bridge between the engineering problems of cloud computing and its business imperatives. Optimisation objectives connect these two concerns, seeking architectures that maximise flexibility, assignments that minimise fragmentation, (re)designs that minimise cost per transaction and so on.

This paper explores this symbiotic relationship between SBSE and cloud computing. Many papers have been written on the advantages of migrating software engineering activities to the cloud (Abadi, 2009; Sotomayor et al., 2009; Armbrust et al., 2010). No doubt there are many advantages of such a migration to SBSE computation *in* the cloud, some of which have already been realised (Le Goues et al., 2012). In this paper we pursue a different direction. Rather than showing that the cloud benefits SBSE (we term this SBSE *in* the cloud) instead this paper explores and develops how SBSE may benefit the cloud (we term this SBSE *for* the cloud). Thus we ask the following question:

> How can SBSE help to optimise the design, development, deployment and evolution of cloud computing for its providers and their clients?

We reach out to two audiences with this paper: first, we aim to encourage SBSE researchers that cloud computing presents relevant research challenges; second, we aim to show cloud practitioners that SBSE has potential to solve many of their optimization problems. The primary contributions of this paper[1] are:

1 To set out a research agenda of SBSE for cloud computing, giving a detailed road map.
2 To make the road map concrete we specify, in detail, five problems within the broad areas of 'SBSE for the cloud'. We reformulate these problems as search problems in the standard manner for SBSE research (Harman and Jones, 2001; Harman, 2007), by proposing specific solution representations and evaluation functions to guide the search.
3 To review how cloud systems pose new challenges and opportunities within existing application areas of SBSE, and in particular search based software testing.

The rest of this paper is organised as follows. First, we give a brief background of cloud computing and SBSE in Section 2. Then we discuss the engineering challenges facing cloud providers and their clients in Section 3. Section 4 gives examples of where and how SBSE can be applied to solve cloud engineering problems, which are new areas of application for SBSE techniques. In Section 5 we outline where the nature of cloud computing offers opportunities for improvement over past SBSE applications. In Section 6, we discuss some common challenges of applying these methods in a cloud scenario before concluding in Section 7.

2. Background

We now provide a brief overview of cloud computing and Search Based Software Engineering.

2.1. Cloud computing

Cloud computing (Mell and Grance, 2009) is a relatively recent and currently high-profile method of IT deployment. Compute facilities such as virtualised server hosting or remote storage accessed via an API, are provided by a *cloud provider* to a *cloud client*.[2] Often a client is a third party company, who will use the compute facilities to provide an external service to their users. For example, Amazon is a cloud provider supplying a storage API to Dropbox, who use the API to provide a file synchronisation service to domestic and commercial users.

In many ways, cloud computing resembles a move away from conventional desktop computing that has been the hallmark of the previous decade, towards a centralised computing model. One stark difference to the mainframes of the timesharing past is that the new paradigm is supported by vast datacentres containing tens of thousands of servers working in unison. These datacentres are described by Barroso and Hölzle (2009) as 'Warehouse-Sized Computers' (WSCs), reflecting their view that the machines can collectively be regarded as a single entity. The coordination of these machines is supported by specialist software layers, including virtualisation technologies such as Xen (Barham et al., 2003) and distributed services such as BigTable (Chang et al., 2008).

Most of the technologies involved in constructing and using cloud computing are not new, but rather it is their particular combination and large-scale deployment that is novel. The emergence of cloud computing would not have been possible without the growth in virtualisation and widespread internet access. Cloud computing promises to commoditise data storage, processing and serving in the way envisioned by utility computing (Rappa, 2004) and service oriented architecture (Papazoglou and van den Heuvel, 2007).

2.1.1. Cloud architectures

Cloud computing is frequently presented as a layered architecture, as shown in Fig. 1. However, in practice these layers are not distinct. For example, data storage services may be regarded as infrastructure, a platform or an application, depending on exactly how those services are being used. We deliberately avoid using these somewhat artificial distinctions.

[1] This is an invited 'road map' paper which forms part of Journal of Systems and Software Special Issue on 'Software Engineering for/in the Cloud'.

[2] In this paper, the word 'user' is reserved for the end-users of software, i.e. the customers of a client. This is complicated somewhat by the fact that some cloud providers serve both clients and users; for example, Google provide App Engine as a service to developers, but they also provide Gmail to users.

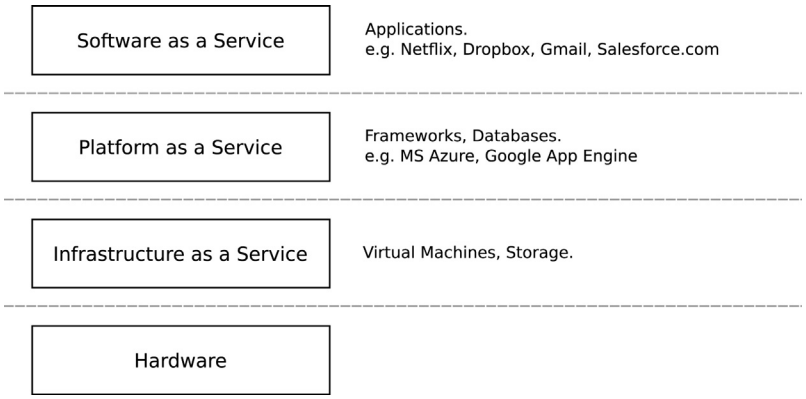


Fig. 1. The cloud stack architecture.

Cloud hardware is typically composed of commodity server-grade ×86 computers arranged in a shallow hierarchy composed of racks and clusters of racks. They are physically located in a series of geographically distributed datacentres. In the case of Amazon Web Services, these datacentres are split into discrete availability zones.

Servers run hypervisor technology such as Xen or VMWare, which manage multiple virtual machines (VMs) on each physical machine. VMs are provided directly to client companies by providers such as RackSpace or Amazon at the infrastructure layer. Each live virtual machine is an *instance* of an offered VM configuration, which is principally defined by the memory and processing power available to the VM. Every instance mounts a *machine image*, also known as an *operating system image*. An image includes an operating system, and the required server software such as web servers or transaction processing software.

The most straightforward example of a platform-level service is Google App Engine, where a customer provides code and Google automatically manages the scaling to respond to incoming requests.

Applications comprise most usage of cloud computing by the general public. Popular current examples include Microsoft Office 365 and Gmail.

2.1.2. Cloud research

Rather than attempting a detailed literature review of cloud computing, we present a simple, informal, graphical overview of cloud research. Fig. 2 is a word cloud generated from the titles of around 25,000 papers returned by a keyword search for *cloud* at the ACM guide to computing literature. The search was performed on 15 May 2012. In this cloud word cloud, the size of a word reflects its frequency in the underlying corpus. Common English words and those occurring rarely are not included in the diagram.

From a cursory inspection of Fig. 2, the largest words are indicative of the topic (*cloud*, *computing*) and the technology (*applications*, *networks*, *web*, etc.). To drill down to the research themes of cloud computing, we remove from the underlying corpus all terms that occur in the concise NIST definition of cloud computing, which is the first sentence of Section 2 in Mell and Grance (2009). Fig. 3 shows the resulting filtered cloud word cloud.

There are two key points to highlight from Fig. 3.

1 As mentioned in Section 1, cloud research issues are not new. The cloud word cloud shows strong links with ancestral research fields including *distributed*, *parallel*, and *grid* computing.
2 The concepts presented in the word cloud are amenable to SBSE optimisation. Apparent terms such as *scheduling*, *clustering*, *reconstruction*, and *optimisation* all admit the potential of SBSE. Terms such as *estimation*, *modelling*, *prediction* and *simulation* show that the cloud research community is attempting to apply well-known existing system optimisation techniques to the cloud domain.

2.1.3. Cloud word cloud in the cloud

As an aside, the cloud word clouds in Figs. 2 and 3 were generated in the cloud. The paper title metadata was collected from the ACM web portal using 50 Amazon EC2 `t1.micro` Linux instances, and stored in a single S3 bucket. The estimated cost of this computation is $0.89, using Amazon Web Services' on-demand prices from May 2012 in the EU (Ireland) region. The graphical representations of the cloud word clouds are produced by Wordle, an online word cloud generation service (Viegas et al., 2009).

This exercise serves to demonstrate the accessibility and economy of cloud computing for simple data processing tasks. We fully expect the rapid, widespread adoption of cloud computing, as predicted by industry analysts IDC (2011).

2.2. Search Based Software Engineering

Search Based Software Engineering (SBSE) is the name given to a field of research and practice in which computational search and optimisation techniques are used to address problems in software engineering (Harman and Jones, 2001). The approach has proved successful as a means of attacking challenging problems in which there are potentially many different and possibly conflicting objectives, the measurement of which may be subject to noise, imprecision and incompleteness. Notable successes have been achieved for problems such as test data generation, modularisation, automated patching, and regression testing, with industrial uptake (Cornford et al., 2003; Wegener and Bühler, 2004; Afzal et al., 2010; Lakhotia et al., 2010b; Cadar et al., 2011; Yoo et al., 2011) and the provision of tools such as AUSTIN (Lakhotia et al., 2010a), Bunch (Mitchell and Mancoridis, 2006), EvoSuite (Fraser and Arcuri, 2011), GenProg (Le Goues et al., 2012), Milu (Jia and Harman, 2008) and SWAT (Alshahwan and Harman, 2011).

The first step in the application of SBSE consists of a reformulation of a software engineering problem as a 'search problem' (Harman and Jones, 2001; Harman et al., 2012a). This formulation involves the definition of a suitable representation of candidate solutions to the problem, or some representation from which these solutions can be obtained, and a measure of the quality of a given solution: an evaluation function. Strictly speaking, all that is required of an evaluation function is that it enables one to compute, for any two candidate solutions, which is the better of the two.

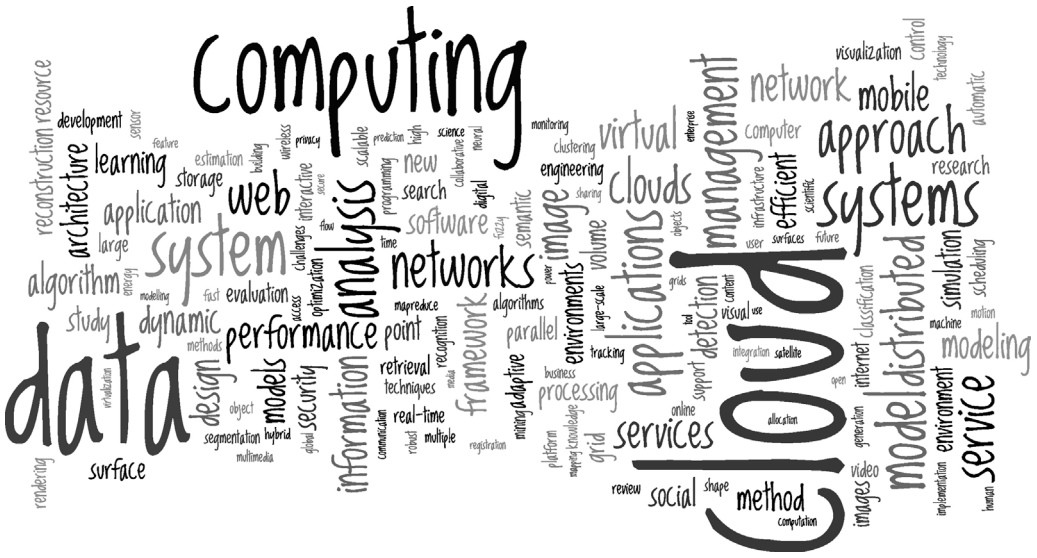


Fig. 2. Cloud word cloud, summarizing 25,000 research papers on cloud computing.

There are many surveys, overviews and reviews of SBSE that provide more in-depth treatment and coverage of results specific to individual engineering subdisciplines such as requirements analysis (Zhang et al., 2008), predictive modelling (Harman, 2010; Afzal and Torkar, 2011), non-functional properties (Afzal et al., 2009), program comprehension (Harman, 2007), design (Räihä, 2010) and testing (McMinn, 2004; Harman, 2008; Afzal et al., 2009; Ali et al., 2010). There is also a tutorial paper aimed primarily at the reader with no prior knowledge of computational search or SBSE applications (Harman et al., 2012b) and a recent bibliometric analysis of ten years' of SBSE literature (2001–2010) (Freitas and Souza, 2011). Fig. 4 shows the recent rapid rise in publications on SBSE.

In this paper we introduce the idea of 'SBSE for the Cloud' by defining representations and evaluation functions for several cloud computing problems. We do this to illustrate how cloud computing problems can be re-formulated as SBSE problems. We seek to provide reasonable coverage of cloud computing applications and issues, but are sure that there are many more such reformulations waiting to be explored.

3. Engineering challenges in cloud computing

The engineering challenges posed by cloud computing can be divided between those facing the providers of cloud infrastructure and those faced by their clients. For the former, there are challenges in the (re-)engineering of their systems to ensure they are suitable for cloud deployment, as well as reducing costs by optimising resource usage. Cloud providers share similar goals in reducing resource usage, but they must also focus on maintaining their service level agreements (SLAs). Fig. 5 details the potential application areas for SBSE methods that we have identified. The figure is annotated with corresponding section numbers for those discussed later in the paper.

3.1. Challenges for cloud clients

From the view of a client, the crucial feature of cloud computing is a reliance on cloud providers for their computational and storage needs. Instead of making capital investment in hardware

Fig. 3. Cloud word cloud, filtered to remove keywords from the NIST definition of cloud computing.

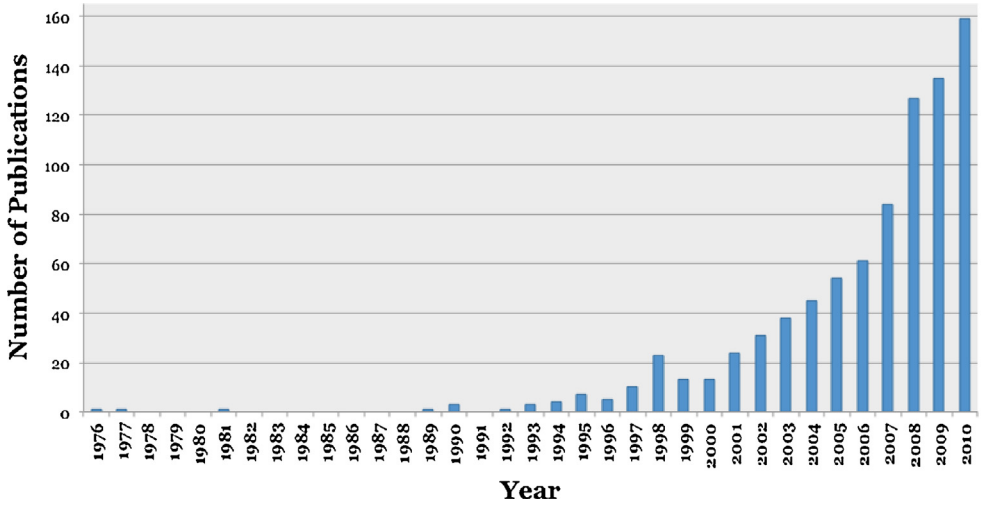


Fig. 4. Yearly SBSE publication rates 1976–2010.

Source: SBSE Repository (Zhang et al., 2012).

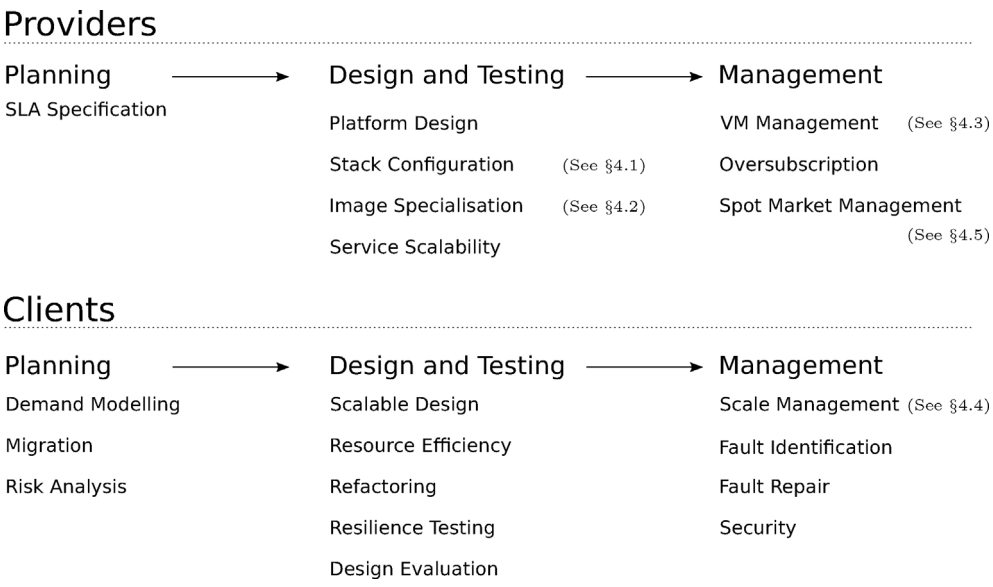


Fig. 5. An overview of the software engineering tasks facing cloud providers and their clients that may be amenable to an SBSE approach.

and hosting their own servers, a company may instead choose to rent facilities from providers on a pay-as-you-go(tm) basis. As with other forms of outsourced deployment, this enables lower overheads through economies of scale and expertise. However, the pay-as-you-go model has additional advantages:

1 Computing expenses are now regarded as an operational overhead, rather than depreciating capital. Thus, large and risk-laden upfront expenditure is not required.
2 The opportunity for scalability is unprecedented. Capacity can be added 'on-demand', without fixed costs and at short notice.
3 There is no economic difference between using a single processor for a hundred hours versus using a hundred processors for one hour. This enables clients to quickly complete tasks that previously would have required longer time periods or substantial capital investment.

Cloud deployment presents a series of engineering and business challenges for cloud clients:

3.1.1. *Predictive modelling*

In order to best optimise cloud deployment through most efficient use of available resources and minimisation of costs involved it will be necessary to predict behaviour, load, use and other profiles that affect resource consumption and costs. Fortunately, there has been much work on predictive modelling and the optimisation of predictive models using search based techniques (Harman, 2010; Afzal and Torkar, 2011) that can be applied to cloud system behaviour prediction.

3.1.2. *Scalability*

Scalability does not come for free. Existing software must be re-engineered to take advantage of the scalability of the cloud. Scalability relies upon parallelism, which is typically provided by either user-level parallelism (many users accessing the same service) or data-level parallelism (data can be processed in parallel). Software must be designed or refactored to have a loosely coupled and asynchronous architecture, to reduce bottlenecks that limit scalability. Furthermore, scalability must be managed;

demand must be anticipated using predictive modelling, and a policy for managing scale must be established.

3.1.3. Fault tolerance

The distributed nature of systems in the cloud and the lower reliability (Vishwanath and Nagappan, 2010) of cloud platforms require a new approach to fault tolerance. Software should be tolerant to complete hardware failure: a virtual machine can be lost at any time, and similarly it should be anticipated that API calls can fail. Expectations of network performance, such as latency, have to be revised. The fragility of the underlying platform is in stark contrast to the expectations of customers using the web services that so often rely on the cloud – such services must have a high level of availability and low latency. Testing procedures must be suitably adapted; the canonical example is Netflix's 'Chaos Monkey' (Netflix Tech Blog, 2008).

3.1.4. Resource efficiency

The cloud pay-as-you-go model makes explicit the variable costs of computation. This provides a greater incentive for companies to be as efficient as possible in their use of resources. The spare cycles of the past are to a great degree eliminated, and therefore decisions such as how and when to scale, which technologies or tool chains to use, and efficient software design, become far more important than they have been during the pre-cloud era. For example, if a company can use a smaller VM instance by requiring less RAM, then it will reduce its operating costs.

3.1.5. Evaluating systems

New scales mean new challenges. Systems that work fine at small scales, or in simulation, may face new problems at larger scales caused by bottlenecks that were not previously apparent. This is prevalent in the analysis of 'Big Data' (Jacobs, 2009). Of particular concern to researchers is that the problem of evaluating new ideas applies to their research as much as, or more than, software development in industry. Problems may only occur at a certain level of scale and concurrency. We discuss this issue further in Section 6.

3.1.6. Security

Maintaining data and system security is a challenge, not least because data that was previously hosted on a company's internal servers may now reside on a third party machine and possibly reside alongside the data and computation of third parties. Traditional firewalls and intrusion detection systems do not apply and the sandboxing of virtualisation limits the visibility of network traffic, impairing the monitoring of this traffic by cloud clients as a defensive measure.

3.1.7. Managing the business model

Reese (2009) notes that it is sometimes lost in discussions of cloud computing that scalability in itself is not an end, rather that scalability must coincide with the goals of the business. For example, serving extra users may not necessarily be profitable, if those users are unlikely to purchase a given product or service. Critical to achieving a sustainable business model is the management of scalability and the predictive modelling of demand.

3.1.8. Accelerated development cycles

The difficulty of distributing software to individual users is often avoided by employing cloud computing. Only server-side software need be updated when changing a service provided over the web, for example. Furthermore, the roll-out of new software can be limited to given users or geographical areas, a technique sometimes referred to as *canarying* or *A/B testing*. Thus traditional barriers to release are no longer an object, and companies such as Google frequently release multiple revisions of their software in a single week. This brings new challenges in an accelerated lifecycle, such as continuous testing, version control and maintaining documentation.

3.1.9. Risk management

There are two major risks facing an organisation hoping to migrate or deploy to the cloud (Armbrust et al., 2009, 2010; Cliff, 2010). The first is the issue of *vendor lock-in*, particularly when making use of platform-as-a-service, which relies on provider-specific APIs. An accepted standard for cloud interoperability has yet to be proposed, and as providers continue to diverge in their approach to provision, such standardisation looks increasingly unlikely. Projects such as Eucalyptus (Nurmi et al., 2009) have suggested a more pragmatic approach, by creating open-source implementations of proprietary APIs. The second major risk surrounds data governance, when data is trusted to a third party (the cloud host) and in situations particular to cloud computing, such as coresidency.

3.2. Challenges for cloud providers

Principally, cloud providers are concerned with maintaining their service level agreements as efficiently as possible, and in a scalable manner:

3.2.1. Virtual machine management

It is convenient to consider the machines that populate a datacentre as a single massively parallel unit. From this point of view, the problems of resource management are clear: scheduling must take place across tens of thousands of machines; individual virtual machine instances must be assigned to physical servers; loads must be monitored and balanced; VMs can be migrated between servers, and memory can be paged across the network (Williams et al., 2011). Coresidency also brings challenges of new pathologies, in the same way that hard disk 'thrashing' may affect conventional shared systems.

3.2.2. Managing oversubscription

Many cloud providers take advantage of the variable workloads of clients by utilising oversubscription to reduce their costs. In an oversubscribed system, the amount of resources provisioned to clients exceeds the total capacity of the physical resources available to the provider. The challenge is to anticipate when demand might outstrip supply, and to take measures to mitigate such situations without violating SLAs.

3.2.3. Translating SLAs to low-level behaviour

There is currently a disconnect between the high-level business model of SLAs and the low-level resource management of kernels and hypervisors.

3.2.4. Scalable service provision

As well as providing infrastructure, most large cloud providers also provide some services, in the form of an API. In constructing such services, they face many of the same challenges as their cloud clients. For example, distributed large-scale storage systems have seen considerable development in the last five years. The main challenge is to ensure scalability and resilience (DeCandia et al., 2007).

3.2.5. Resource efficiency

Providers aim to reduce their expenditure through optimising the consumption of resources including CPU time, memory

usage, data storage space and power consumption. For example, modern datacentres are for the most part power-inefficient (Barroso and Hölzle, 2009). The main overhead of running a datacentre is climate management. Most effort in this area focuses on the design of cooling systems, but once efficiency improves, the focus may shift onto 'energy-scalable' computing, that is ensuring that low utilisation corresponds to low power consumption and vice-versa.

4. Example applications of SBSE to cloud engineering

We now provide illustrative examples of specific optimisation problems in cloud computing, in order to demonstrate the applicability of SBSE to cloud engineering. To avoid imprecision, we give detailed suggestions on how each task may be formulated as a search problem, and also reference related work in SBSE that could be applied in this domain. Detailed descriptions of SBSE algorithms and the software engineering applications to which they have hitherto been applied can be found elsewhere in surveys on SBSE (Harman et al., 2012a).

We have chosen optimisation problems that can potentially benefit both, cloud providers as well as their clients. The examples given in Sections 4.1, 4.3 and 4.5 address specific problems faced by cloud providers of Infrastructure As A Service, Platform As A Service or Software As A Service. The problems and solutions laid out in Sections 4.2 and 4.4 can serve to benefit both, cloud providers as well as clients.

4.1. Provider resource efficiency: cloud stack configuration

One challenge to both providers and clients, noted in Section 3, is to reduce the resource consumption of their systems.

In this section, we propose an approach that will help cloud providers optimise their cloud stack configuration of physical servers inside a datacentre. Fig. 6 illustrates a typical server software architecture in a cloud environment. A physical server in a cloud datacentre executes a hypervisor, generally Xen or VMware. The hypervisor runs a single operating system, such as a modified version of Linux, as the *dom0* host operating system. This host is controlled by the cloud provider to administer the node; residing in dom0 provides it with special access privileges. Guest operating system instances are the virtual machines utilised by a cloud client, and execute within *domU*. These are multiplexed on the node by the *dom0* host. Each of these layers may be configured independently.

There are many opportunities for optimisation: both the dom0 host and the domU guest operating systems can be subject to configuration tuning, in order to optimize particular performance characteristics, such as network bandwidth and energy efficiency.

Cloud providers use host configuration parameters to offer a range of instance configurations, each with fixed characteristics. For example, the Amazon `t1.micro` instance currently offers 613 MB of RAM and up to two virtual processing units. These characteristics may map directly onto the appropriate Xen configuration file parameters:

```
maxmem = 613
vcpus = 2
```

Such *host* configuration files specify how the physical resources of the underlying hardware node should be divided amongst guest operating systems. This configuration can even be altered dynamically.

The *guest* operating system settings are generally configured at boot-time, for example through the boot parameters to the Linux kernel (Red Hat Enterprise, 2007). Many of these parameters are performance-sensitive. Some parameters can be configured at runtime, but even changes that require a reboot may be applied in a distributed and robust cloud solution where shorter uptimes are anticipated.

4.1.1. Formulation

We can represent the parameters of the hypervisor or operating system as a value vector containing multiple types, suitable for the application of a hill-climber or genetic algorithm. Search operators should respect parameter boundaries and constraints that determine the structure of feasible solutions. In an offline scenario, in which the optimisation is performed in a safe sandbox environment disconnected from the live cloud instances, this may be achieved by punishing unfeasible solutions using a specially designed fitness function. In online optimisation, however, the search algorithm will evaluate the fitness of candidate solutions *in situ* using the live cloud instances as the fitness function. Consequently, the search will have to be limited to the subspace of feasible solutions.

In general it seems reasonable to assume that an individual user's previous workload patterns may be indicative of future behaviour, and optimise accordingly. This approach would be similar to the work on SBSE for compiler optimisation, which searches the space of parameter settings: SBSE has been advocated as a tuning mechanism for compilers (Cooper et al., 1999). Hoste and Eeckhout (2008) demonstrated how a compiler can be tuned by targeting different performance objectives for the code it produces. Compilers have a surprising number of possible parameters that can be tuned in this manner: `gcc` has about 200. Indeed, one way to tune an operating system or other cloud stack component would be to search for suitable compiler settings that enable a recompiled version to perform better in a given scenario.

4.1.2. Evaluation function

The evaluation function for the tuning of cloud stack components would depend on the performance objectives to be monitored and optimised. Obvious choices are dynamic performance attributes including memory usage, execution time and energy consumption. Since these properties are subject to variability depending on the given workload, the evaluation function will be forced to seek normalisation through sampling and statistical analysis. This is a cross-cutting issue in cloud optimisation and is discussed further in Section 6.

4.2. Client resource efficiency: image specialisation

Cloud providers usually find themselves running many instances using the same machine image, the full features of which are unlikely to be required in every case. On a given virtual machine, a customer will usually be running a specific application, such as a LAMP (Linux-Apache-MySQL-PHP) stack. Either through explicit declaration of the intended use by customers, or by permitted monitoring of activity, the provider could tailor the image to better fit the needs of the customer.

It seems wasteful to use only a fraction of software within an image; the larger the scale of the datacentre, the worse this problem of 'unused software plant' will be. Specialisation could help in reducing image size, reducing execution time, and lowering the time required to migrate an image across the network (Voorsluys et al., 2009). The beneficiaries of such specialisation would be both, cloud providers as well as cloud clients. For a cloud provider it will result in less demand on physical hardware. For cloud clients, specialisation offers one way of cutting cost, for example by consuming less resources.

At present cloud providers already offer many different virtual machine images. For example, Amazon currently provides 755

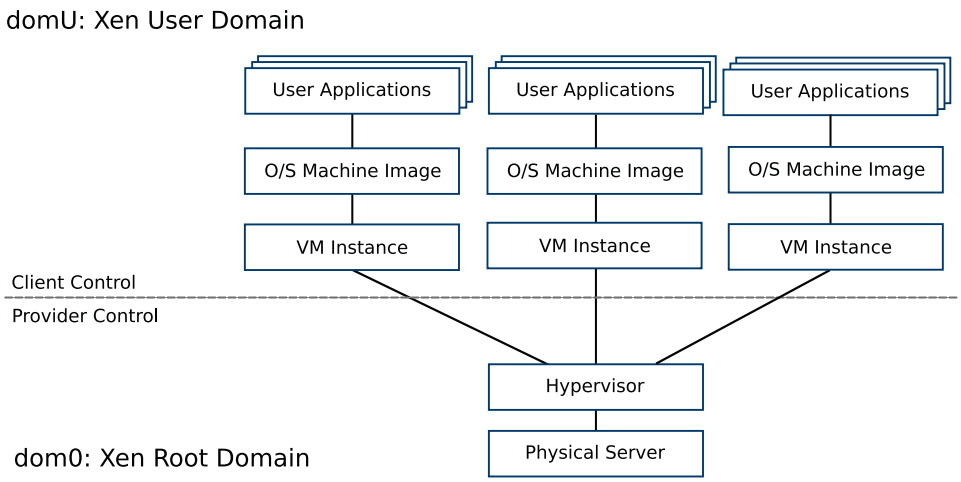


Fig. 6. Schematic diagram of cloud application execution stack.

images[3] for customers on the EC2 cloud platform. Each image includes a particular operating system release and a set of bundled applications. The 'BitNami WordPress Stack' image provides Ubuntu Linux and all necessary software to support WordPress: BitNami, WordPress, Apache, MySQL, PHP and phpMyAdmin[4]. Such coarse-grained specialisation can be formulated as the selection of a set of appropriate Linux distribution packages (e.g. RPMs) to include in a VM image. There is a well-defined dependency graph structure for RPM packages, which could be co-opted to create specialised Linux images for the cloud.

There exists the interesting problem of identifying the trade-off between the frequency of use of a module versus potential size reduction should it be removed. If removing a rarely used module can result in a significant space saving, a workaround may be worth considering.

More fine-grained VM specialisation is also possible. A single RPM (e.g., a Linux library such as `libstdc++`) may contain superfluous functionality for the purpose of a given VM instance. Program slicing might help to reduce the library to the bare essentials for the use cases required (Guo and Engler, 2011). Even more ambitious specialisation may be achieved by rewriting application and operating system components. Manual specialization effort (Madhavapeddy et al., 2010) demonstrates clear potential; the application of SBSE should automate this process.

4.2.1. Formulation

For the reduction of image size through the deletion of unused modules, the search could use a dependency graph representation. Searching for compromises between usage and size reduction is closely related to the cost-benefit trade-offs in requirements engineering (Zhang et al., 2007; Durillo et al., 2009).

More generally, this is a type of 'partial evaluation' of the machine image with respect to the intended application. Partial Evaluation (PE) has been studied for many years as a way of specialising programs to specific computation (Beckman et al., 1976; Bjorner et al., 1987; Jones, 1996). However, highly non-trivial engineering work would be required to adapt and develop existing PE approaches to the scale required, such as modifying the Linux kernel, which may run to several millions of lines of code.

Another approach would be to use program slicing (Harman and Hierons, 2001; Silva, 2012) to 'slice away' unused parts of an image that can be statically identified. Like partial evaluation, slicing has been studied for many years (Weiser, 1979). However, despite many advances in slicing technologies, scalability is currently limited.

An alternative to PE and slicing for image specialisation would be to use a search based approach to find parts of the image that can be removed. More ambitiously, a search based approach might also seek to construct a new version of the image, specialised for a particular application, using the original image to provide both a starting point and an oracle with which to determine the correct behaviour of the re-write. A natural choice of formulation for this problem would rest on the use of genetic programming. Related SBSE work using Genetic Programming (GP) (Koza, 1992) includes recent advances in automated solutions to bug fixing (Arcuri and Yao, 2008; Weimer et al., 2009), platform and language migration (Langdon and Harman, 2010) and non-functional property optimisation (White et al., 2008, 2011; Sitthi-Amorn et al., 2011).

A typical representation in a GP search is an abstract syntax tree. However, the GP process inherently involves maintaining many copies of the program under optimisation and would hence be memory-intensive. Fortunately, recent applications of genetic programming to the problem of cloning systems and subsystems have developed constrained representations to overcome such scalability issues. For example, Langdon and Harman (2010) use a grammar based GP approach, in which changes are constrained by a grammar specialised to the program, while Le Goues et al. (2012) represent a solution as a sequence of edits to the original in a manner reminiscent of earlier work on search based transformation (Ryan, 2000; Fatiregun et al., 2005). Either approach could be used to scale GP to the image specialisation problem. Profiling could also focus the search, identifying those software components that are frequently executed.

Sufficient knowledge of a system's behaviour will play a critical role. The upper limit of the deletion approach is defined by the minimal functionality that must be preserved. It is the behavioural characteristics that would provide a pseudo-specification for the specialisation; the resulting software should be able to handle all system behaviours that clients are interested in. The primary source of this knowledge should be usage profiles.

Accumulated profile data would provide a certain level of confidence in determining the behavioural characteristics that must be retained. If further confidence is required, Search Based

[3] https://aws.amazon.com/amis (accessed April 2012).
[4] https://aws.amazon.com/amis/bitnami-wordpress-stack-3-3-1-0-ubuntu-10-04.

Software Testing (SBST) (McMinn, 2004; Harman and McMinn, 2010; Lakhotia et al., 2010a; Alshahwan and Harman, 2011; Fraser and Arcuri, 2011) could also be used to specifically target these apparently unexecuted regions of code to raise confidence that they are not executed. Code that resists execution for the application in question, even when search is used specifically to target its execution, might be speculatively removed.

4.2.2. Evaluation function

Natural target metrics are image size, memory usage, and execution time. For image size, we may consider the trade-off between the size of images and the functionality provided. Image size is static, eliminating the need to make repeated observations. With regards to memory usage and execution time, we may consider whether average, peak or minimal requirements are most important. The average draw on either of these resources may not be as important as the variance, and we might seek to raise the minimum draw on resources and reduce the maximum draw, thereby aiding load prediction and management on a cloud infrastructure.

If an online approach is adopted, then an engineer might also be presented with occasional reports of Pareto-optimal trade-offs (Harman, 2007) currently possible for heavily used applications. In all optimisation work, it is important to consider the appropriate point at which to deploy human judgement in an otherwise automated process (Harman, 2007, 2010). These human judgements could be informed by the automated construction of speculative optimisation reports from realtime image specialisers, running concurrently with the applications they serve and which absorb and make use of otherwise redundant datacentre resources to optimise for the future.

4.3. Virtual machine assignment and consolidation by providers

Virtualisation offers opportunities for power consumption reduction using consolidation, whereby some virtual machines may be migrated from one physical server to another. Traditional consolidation can be employed so that some servers may be powered down or transitioned to a low-power configuration. Servers continue to use significant amounts of power when they are idle; for example, Google report that idle power is 'generally never below 50% of peak' (Fan et al., 2007). Hence consolidation may be more desirable than load-balancing, although powering down servers has an associated cost in the time taken to restore them.

The problem of allocating VMs to servers is essentially one of bin-packing, albeit dynamic in nature and under many constraints. The constraints are implied by the SLAs between the cloud provider and its clients. For example, we may seek to make power savings as described above, but also wish to avoid cohosting two VMs from the same customer on the same machine or network segment, so as not to affect the terms of an SLA in regards to availability or responsiveness. To complicate matters, it is common practice to oversubscribe physical hardware (Williams et al., 2011), such that demand may in some circumstances be expected to exceed supply. Furthermore, the energy profile of an application and of a server appears to be highly variable and experimental studies investigating servers' energy consumption have reported conflicting findings (Barroso and Hölzle, 2009; Lee and Zomaya, 2012).

The allocation of VMs to servers has two overriding goals: firstly, to reduce energy consumption by shutting down unoccupied machines when demand is low within a datacentre, i.e. traditional 'consolidation'. Secondly, to make efficient use of machines that are running without impacting on a customer's VM in a way that SLAs cannot be met. Aggressive consolidation may lead to competition for resources on heavily loaded servers, and it may also be problematic if demand rises faster than powered-down servers can be restored. Hence this problem and the predictive modelling of demand are related.

There exists a substantial body of literature on virtual machine management (Srikantaiah et al., 2008; Beloglazov and Buyya, 2010; Lee and Zomaya, 2012), which invariably suggest handwritten heuristics. The application of SBSE methods offers an opportunity to increase the sophistication and efficiency of management.

4.3.1. Formulation

Let us first consider the simplest case of a single static situation. We may express this task as a search problem by considering a solution as a mapping from a description of the demand for resources onto an existing infrastructure of physical machines. To build on the formulation from Lee and Zomaya (2012), a servers s_i out of w servers belongs to the set S, and the utilisation U_i of a single server $s_i \in S$ can be described as the sum of the utilisation $u_{i,j}$ due to each resident virtual machine v_j running on s_i thus:

$$U_i = \sum_{j=1}^{n} u_{i,j} \tag{1}$$

The energy consumption of a single server, c_i, must then be modelled as a function of its utilisation,

$$c_i = b + f(U_i) \tag{2}$$

where b indicates the minimal (base) energy usage of a server that is idling. However, this assumes that all utilisation is homogeneous, whereas in fact we might regard U_i as a k-dimensional vector, giving utilisation for each of the k types of resource (CPU, memory, energy, etc.) provided by a server. Lee and Zomaya also assume that f is a linear function, which is in conflict to that reported by Barroso and Hölzle (2009).

Momentarily discarding these limitations, the problem can then be considered as searching for an assignment consisting of a vector A such that A_j describes where virtual machine v_j should reside. The goal is to minimise overall energy consumption, C:

$$C = \sum_{i=1}^{w} c_i \cdot R_i \tag{3}$$

where R_i gives a boolean value indicating if a server is running at least one virtual machine:

$$R_i = \begin{cases} 1 & \text{if } \exists j : A_j = i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

It is not unreasonable to assume that a provider may wish to separate virtual machines from the same customer onto separate servers, i.e. letting $\text{cust}(v_x)$ denote the customer who is running virtual machine v_x, we want to enforce the following constraint:

$$\forall i, j, i \neq j : \text{cust}(v_i) = \text{cust}(v_j) \Rightarrow A_i \neq A_j \tag{5}$$

There will be further constraints based on the risk profile of highly utilised servers. Furthermore, we may consider the costs of transition from the current allocation A_t at time t to the suggested allocation A_{t+1}. If we assume a fixed cost for VM migration, then we may try to minimise the difference between A_t and A_{t+1}. If a server is to be shutdown or restarted, then we must take the associated cost into consideration.

Much depends on whether the problem is treated *statically*, in the sense that a single allocation is found for a given situation, or whether the goal is to construct a policy to be executed *dynamically*. One interesting hybrid approach would be to search for a static allocation that not only optimises a given evaluation function, but also takes into account future scenarios, for example in order to minimise the cost of migrations given potential anticipated

demand changes. This idea is similar to that of Emberson and Bate (2010) in their SBSE work on allocating processes to processors in a multiprocessor system.

The static allocation of VMs to physical servers can be represented using the integer vector, A. A dynamic solution will be represented as a program or heuristic function

$$A_{t+1} = f(A_t, U_t) \quad (6)$$

A more ambitious policy could incorporate more complex information, such as anticipated demand, into this function. The above discussion is heavily simplified, and the constraints involved will depend on the design of the datacenter, as well as the goals of the parties involved. Justafort (2012) considers the impact at application level, for example, where Stillwell et al. (2012) optimise on heterogeneous platforms.

Searching over a vector space could be performed using Simulated Annealing (Kirkpatrick et al., 1983), the technique used by Emberson and Bate.

The search for a dynamic heuristic would be well-suited to a Genetic Programming approach, with a function mapping virtual machines to physical servers represented as an expression tree. Previous work on scheduling in GP provides examples of how this might be achieved (Jakobović and Budin, 2006; Jakobovic et al., 2007).

4.3.2. Evaluation function

The objectives of this task are to minimise power consumption within the constraints specified by an SLA whilst taking into account potential future usage. SLA constraints will include providing enough physical RAM, ensure sufficiently low latency, and providing enough independence of execution to maintain uptime guarantees – for example, by avoiding the coresidency of a client's own virtual machines on a single server. Future usage must be taken into account to ensure that violation of an SLA constraint is unlikely to occur after reassignment.

There are three approaches to evaluation the quality of a solution: modelling, simulation and physical evaluation; the latter can involve real-time feedback from the datacentre. The CloudSim toolkit (Calheiros et al., 2011) provides an appropriate level of abstraction for this evaluation function and was used previously by Beloglazov and Buyya (2010).

4.4. Scale management for cloud clients

One of the key advantages of cloud computing is the ability to automatically scale a server infrastructure for an application in response to changes in demand. A cloud client will wish to minimise their expenditure; they are billed for the computational resources that they consume. A higher specification VM, with more RAM or a faster processor, incurs a higher cost. Further, costs are typically based on 'instance hours', i.e. how long a VM is running for, regardless of the *amount* of computation performed. Idle instances cost money and it is important to bring more VMs up when there is a surge in demand, and scale an application down by terminating VM instances when demand drops. As noted in Section 3.1, this scalability must be managed.

Thus a cloud application must be configured such that it minimises resource usage while maintaining a certain level of throughput. This can be achieved by either anticipating demand or by constructing a set of rules that react to changes in an application's usage profile. Fluctuations can be caused by factors including promotional campaigns, the 'Slashdot effect' where a popular website links to a smaller website, and even behavioural patterns of users. An example often cited is the electricity spike caused by kettles switched on at half time during popular sporting events.

Some variations in demand are predictable, while others are not. Cloud providers Amazon and Microsoft Azure allow a client to configure rules for scaling an application. These rules are based on metrics including maximum, minimum and average CPU usage, the number of items in a servers' request queue and so on. They are typically designed by a developer and system administrator.

Fig. 7 shows an example snippet from a Microsoft configuration document. The top half contains rules for a scheduled scaling of an application (also called 'constraint rules'), while the bottom half shows rules that define when to react to fluctuations in an application's usage (also called 'reactive rules'). Both types of rules should be configured in a way that maximises performance, but minimises cost. For example, one would not want to scale beyond the level specified by the constraint rules, because otherwise a client will be charged for idle instances. Equally, the reactive rules need to be configured such that they are robust to sudden spikes in user demand. We can consider using SBSE to optimise such rules in order to find an optimal trade-off between performance and cost.

4.4.1. Formulation

The rules shown in Fig. 7 are described in an XML format. As such they are already in a format amenable to a GP approach. The structure of an XML document is defined in different schemas that can be used by the search to ensure it only generates valid XML documents.

Rules are composed from conditions and associated actions. Condition operators such as `equals, greater, greaterOrEqual, less, lessThan, not` can be used to form part of the function set. When a condition evaluates to `true`, the corresponding action is performed. Actions are also functions that take a variable number of parameters. For example, the `scale` action is parameterised by the type of VM to scale, as well as how many replicated VMs should be created.

The terminal set can be constructed from the attributes defined for the different elements in the XML schema, along with their values. For example, a `startTime` attribute is of type `timestamp`. In addition, the terminal set needs to contain all possible VM configuration options along with the data types required by the function set (e.g. the range of integers).

4.4.2. Evaluation function

We can treat this as a multi-objective optimisation problem where our objectives are *latency* and *cost*. Assume we have an existing load test suite. We may evaluate a given set of constraint and reactive rules by measuring the average latency observed for the load tests. Equally, we can measure the cost of executing a load test suite in terms of computational resources required.

Evaluating a set of scaling rules by running an entire load test is likely to be too computationally expensive in practice. Forrest et al. (2009) showed in their work on automatic patch fixing that sampling from a set of test cases can be more efficient than using an entire test suite. In their work, test cases were used to validate candidate patches, and a patch was considered valid if it passed all test cases. They found that a sampling method provided sufficient information to a GP in order to guide it towards possible solutions. Candidate solutions only have to be evaluated on the complete test suite once an optimisation run finished, or after a given number of generations.

Since the evaluation function uses two conflicting objectives, the GP will not generate a single solution. Instead we will obtain a *Pareto front*. A Pareto front is a set of *non-dominating*, Pareto optimal, solutions. Solutions are said to be *non-dominated* if no other solution exists that is better in all objectives i.e. both latency and cost.

Since the output of the search will be a set of equally good solutions, a decision maker has to decide which solution to pick.

```
<constraintRules>
    <rule name="Peak" description="Active at peak times" enabled="true" rank="100">
        <actions>
            <range min="4" max="4" target="RoleA"/>
        </actions>
        <timetable startTime="08:00:00" duration="02:00:00">
            <daily/>
        </timetable>
    </rule>
</constraintRules>
```

Example of constraint rules

```
<rule name="Example Scaling Rule" rank="100">
    <when>
        <greater operand="CPU_RoleA" than="80"/>
    </when>
    <actions>
        <scale target="WorkerRoleA" by="2"/>
    </actions>
</rule>
```

Example of reactive rules

Fig. 7. Example configuration rules for when to scale an application. Snippet taken from a Microsoft Azure configuration file (Microsoft, 2012). Such rules can be generated and optimized using genetic programming.

Visualising a Pareto front in two dimensions can help with this process as it illustrates the trade-offs between lower latency and cost.

4.5. Spot-price management

We have seen how SBSE might be used to reduce the cost for cloud clients by evolving optimised scaling rules for an application. Now we examine the management of demand from the position of a cloud provider.

From the point of view of a cloud provider, when a server is not running at full capacity, underutilisation of resources equates to lost potential revenue. Thus, it is important to maximise the usage of their cloud infrastructure at any given time, given sufficient spare capacity to cope with anticipated demand surges. One way to address this problem is to *auction* off unused resources, such as Amazon's Spot instances (Amazon, 2012). A client can bid for a spot instance, and when their bid matches or exceeds the price of a spot instance, the instance becomes available to them. A spot instance remains available to the client for as long as their bid price matches or exceeds the price of an instance. Prices are periodically adjusted by the cloud provider, and when a spot price exceeds a client's bid, they lose their instance.

To maximise the effectiveness of spot instances, a cloud provider must efficiently divide spare capacity amongst suitable instance configurations, and subsequently make those instances available for auction.

4.5.1. Representation

We propose to formulate the problem of auctioning off unused capacity as a bin-packing problem, in a similar manner to the consolidation problem in Section 4.3. Bins denote servers with spare capacity and the objects to place into the bins are VM instance types. We assume that a provider only offers a limited number of spot instance sizes. The goal is to distribute VM instances across a server infrastructure such that the amount of idle server resources are minimised. For ease of explanation we will only consider the distribution of VM instances over a fixed time period of one hour.

Let us denote the different types of VM instance a provider may offer as $vt_1, \ldots, vt_n$. We can simply represent the possible allocation of VM instances to server bins as a matrix, as shown in Fig. 8. Rows in the matrix denote VM instance types and the columns denote server bins. The numbers in the row/columns denote how many new instances of a particular VM type are allocated to a specific server.

4.5.2. Evaluation function

As an evaluation function we propose to minimise spare capacity. More formally, let h_i denote the spare capacity (headroom) of server s_i. This is determined with reference to Eq. (1):

$$h_i = 1 - U_i = 1 - \sum_{j=1}^{n} u_{i,j} \tag{7}$$

We must then find a new allocation of spot instances to servers, represented by the matrix B as illustrated in Fig. 8 where and $B_{i,j}$ denotes the number of new VM instances of type j, to be deployed on server i.

The evaluation function will be to reduce h_i over time, under the constraints implied by SLAs. Offering the VMs according to the matrix B (whether found directly or output by a search-generated policy) is not enough to improve utilisation, as buyers for the new VM instances must be found. In particular, B does not generate a price at which a particular VM type should be auctioned off.

Ben-Yehuda et al. (2011) showed that spot instance prices appear to be set, not by supply and demand, but rather a randomised algorithm. In particular, the authors found the following formula to be a good fit for historical price setting:

$$P_i = P_{i-1} + \Delta_i \tag{8}$$

where $\Delta_i = -0.7 \times \Delta_{i-1} + \epsilon(\sigma)$. In their formula, $\epsilon(\sigma)$ denotes white noise with standard deviation σ. The authors matched σ to a value

	(600MHz,128RAM)	(1200MHz,512RAM)	(2400MHz,256RAM)
s_1	3	0	0
s_2	0	1	0

Fig. 8. Example matrix representing possible VM instance type allocations to server bins.

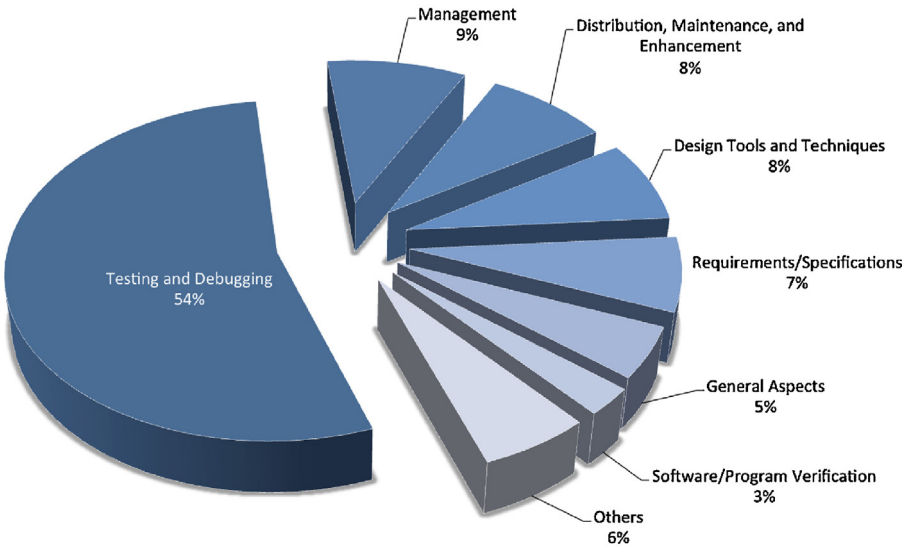


Fig. 9. Share of papers targeting each SBSE research application.
Source: SBSE Repository Zhang et al. (2012) (data from 1976 to May 2012).

of $(0.39 \times (C - F))$, where C denotes the maximum (ceiling) and F denotes the minimum (floor) price of a particular VM instance. Once a search algorithm has found an optimal allocation of VM instance types for idle resources, the VM images can be pre-built and sold off. The price for an instance type can then be set using the above formula.

5. Challenges and opportunities for existing SBSE methods

In the previous section we presented an overview of challenges posed by cloud systems that will lead to *new* applications and objectives for the SBSE research and practitioner community. In this section we turn our attention to existing areas of SBSE research that face new challenges and new opportunities when applied in the cloud domain.

Search Based Software Engineering research has produced many distinct subdisciplines, most of which will remain comparatively untouched by the migration to cloud platforms. For instance, the problems of requirements engineering (Cheng and Atlee, 2007) and the associated SBSE research (Zhang et al., 2008), will not necessarily change fundamentally. The main challenge is likely to remain the elicitation, understanding and balancing of multiple requirements and their relationships and tensions, in the presence of multiple competing and conflicting stakeholders, with poorly understood and ill-expressed needs and desires.

However, for other areas of SBSE research and practice, there will be a significant change, when the application focus moves from traditional platforms and business models to a cloud paradigm.

5.1. Search based testing for the cloud

One of the biggest changes can be expected in the area of SBSE for testing, known as Search Based Software Testing (SBST) (McMinn, 2004; Harman, 2007; Ali et al., 2010; Harman et al., 2012a). The changes in this area are not entirely problematic; cloud platforms offer significant advantages to the tester that researchers in SBST may seek to exploit. This is particularly significant, because testing and debugging form approximately half of the overall research activity in SBSE (see Fig. 9). As such, an effect on SBST will have a significant impact of the SBSE research agenda as whole.

5.1.1. On-demand test environments

One of the major bottlenecks in testing is the time it takes to prepare test environments and execute test cases. Even when tests can be executed in parallel, limited availability of hardware means overall test execution time remains high. Further, up until now, it has not been easy to setup a test infrastructure to run tests in parallel. Cloud platforms offer a new solution to this problem. Apart from unit tests, many tests such as integration and load tests depend on an execution environment. We can easily replicate such environments by cloning a VM image. These can then be distributed across many virtual machines to run in parallel.

5.1.2. Snapshotting

In a cloud environment, every application runs inside a virtual machine. Thus, it is possible to take a snapshot of an application, including its entire runtime environment, at an arbitrary stage in its execution. This opens up many new possibilities for debugging.

Traditionally software is deployed and installed on client machines. Most software companies have little or no control about the execution environment of their application.[5] This lack of information, or rather the huge configuration space applications run within, means debugging a failure is a difficult process. In cloud applications, the specification of a user's machine is likely to have much less impact on the correct running of software; users' machines are becoming more and more akin to dumb terminals. Since an application resides in the cloud, developers have full access and control over its environment. Thus, when a failure occurs, we are able to take a snapshot of the entire VM state for debugging purposes.

In a similar manner, there is the possibility of rapidly forking a running VM instance (Lagar-cavilla et al., 2009). This will further enable efficient software testing, by enabling a search process to bifurcate the execution of a problem based on, for example, a branch predicate.

5.1.3. Multi-version deployment

One way to test for regression faults, or evaluate patches, is to deploy modified software only to a certain group of users, known

[5] For example some Windows applications are deployed under Linux with the help of third party software such as Wine (2012).

as *canarying*. This notion is similar to alpha testing new features. For example, Facebook recently evaluated its pay per post feature on a relatively small audience in Australia and New Zealand, while the rest of the world continued to use the 'old version'. This means we are able to easily collect profile information across versions. We may use this information to help SBST, for example through seeding of optimisation algorithms.

5.1.4. Quantifiable cost

It is also worth noting that the cloud offers a new way of evaluating the cost of testing. Le Goues et al. (2012) used the cloud to measure the cost of a bug fix. This is simply the monetary charge incurred for the computational resources used, such as the time it took to find a patch. In testing we could equally measure the cost of code coverage, the cost of finding a fault and so on. Just as a cloud client might consider whether scaling to another VM will bring sufficient return in revenue, we may also consider the return on investment of additional resources applied to testing. The elastic resources of the cloud must be used efficiently.

5.1.5. Short release cycles

Previously, bugs were expensive to fix once software had been deployed. This was partly because software was 'shipped', and thus it was not easy to patch deployed software applications. Even when patches were available for download online, no assumptions could be made whether a patch had been downloaded and installed by a user or not. This in turn lead to its own security vulnerabilities, since hackers could use patches to detect security vulnerabilities, and then exploit users that had not updated their software with the latest patch.

Cloud computing also offers new opportunities in this area. Software no longer needs to be shipped, as it is provided in the cloud. Thus, when a VM is updated with a patch, all users of an application automatically start using the patched version, without their knowledge or intervention.

The ability to easily deploy software is leading to ever shorter release cycles. Small changes can be pushed to every user of the software, almost in an on-demand manner. These short release cycles will require improvements in how regression testing is performed. One research avenue could be how to make regression testing fit into a cloud development cycle. It needs to be efficient, so it does not counteract the notion of 'short release cycle', while remaining robust.

5.2. Software maintenance

Much previous work in the area of software maintenance within SBSE has actually focused on refactoring software. For example, a common goal has been to refactor software to achieve a given quality in terms of cohesion or coupling metrics (Lutz, 2001).

As cloud computing relies on a centralised system of software distribution, it presents new opportunities for automating maintenance through search that were not possible in a standard desktop computing scenario. Firstly, software changes can be rolled out to users without user intervention. Secondly, these updates can be rolled out to a limited audience through canarying to limit the impact of any erroneous modifications, as well as the facility to rapidly backout those changes should it be required. This could potentially allow SBSE researchers to automate more of the process than has previously been palatable, as the risk profile of software updates is reduced.

Running many virtual machine instances, potentially duplicating software across many machines, offers new opportunities for fault-finding and repair. Snapshotting, as described above, allows improved fault localisation by effectively capturing the information required to recreate a bug, or place the system in a state where a bug may be replicated given the correct inputs. In addition, it opens up the possibility of attempting to repair the problem automatically, particularly if the problem relates to performance or resource consumption.

For example, consider the work of Carzaniga et al. (2008), who used different paths through a system, such as similar API calls, to find a workaround for a given problem. Their work relied on the redundancy inherent in software in that the same functionality can often be found in multiple locations. A similar approach can be followed using different versions of the same software, and those versions can also be used as oracles when automating repair. In the cloud, there may exist many different versions of the same software, all available to a repair mechanism through a global API. Similarly, we may use snapshots to roll-back, modify, and re-run a system after a crash or performance problem. An obvious method of repair is to vary the software's environment, by using a different machine image or physical location in the datacentre.

6. Cross-cutting issues

This section discusses issues that must be addressed in the design and implementation of experiments investigating SBSE for the cloud. Researchers and engineers will encounter three common themes: the problem of effective evaluation of their ideas or designs, how best to sample the available data in order to make their decisions, and the phenomenon of increasing specialisation.

6.1. Prototyping and evaluation

Cloud datacentres are typically composed of servers in the tens of thousands, and cost hundreds of millions of dollars to construct. Constructing a datacentre is economically infeasible for an academic institution, but we must be able to evaluate our research with sufficient fidelity to enable us to establish their credibility. Engineers working in the industry face similar problems, and may be limited to using existing facilities when demand for them is low, or evaluation using canarying: limited roll-out of changes to a restricted audience. If neither of these options are available, they face the same challenges as academic researchers.

When working at the application level, for example when carrying out resilience testing, existing cloud services can be used. However, any work that requires access to the lower levels of the cloud stack will need a cloud testbed. The first option is to use simulation, and there are a number of cloud simulation tools under development (Kliazovich et al., 2010; Calheiros et al., 2011). Such simulators may be appropriate for evaluating course-grained behaviour under a given VM management system, by simulating migration and consolidation patterns without concern for detailed application behaviour.

The efficiency and fidelity of simulation restrict its application. Therefore, it seems that part of the research agenda into cloud computing must be the evaluation of a 'scale model' cloud approach. Such small-scale cloud models could be constructed by building fragments of cloud systems (e.g. a single rack of servers), or by replicating larger slices of a datacentre using less capable hardware.

In designing both simulations and scale models of real cloud systems, there will be natural open research questions about whether the models correctly simulate the behaviour exhibited by the real world cloud. This is not an issue that is specific to SBSE for the cloud, but an issue for any work that seeks to evaluate and experiment with a model or simulation of the cloud and, indeed, is an issue for any model of any system, such as for example, models and simulations of financial systems (Cohen et al., 1983; Panayi et al., in press).

These issues are challenging but they are not insurmountable. In the financial modelling domain for example, Darley and Outkin (2007), were able to construct a very effective model of the markets which was used to predict the impact of Nasdaq stock market decimalisation. If complex dynamic systems such as financial markets can be effectively modelled and simulated, then there is grounds for hope that the same may be true of cloud models.

Furthermore, the issue of designing a suitable cloud model is, itself, an optimisation problem, making SBSE an ideal candidate for cloud modelling too. The optimisation objective is to design the model (selecting parameters, architecture and properties) such that predictions and behaviours observed using the model are appropriate simulations of real world clouds. This approach to design and tuning of models and simulations using search based optimisation has already been applied to financial models (Rogers and von Tessin, 2004; Narzisi et al., 2006). There is no reason to assume that it cannot also be applied to optimising the design of cloud models and simulations.

The further challenge of generating representative workloads is also problematic. In particular, we might hope to replicate realistic workloads as seen in existing datacentres. Due to the multi-residency of datacentres, this data may not be publicly available. Some providers have released limited data (Mishra et al., 2010; Google, 2012).

6.2. Monitoring and sampling

When discussing design issues at the high level, the difficulty of measuring properties that are essential components of an evaluation function can be lost. For example, consider power efficiency: not only is it difficult to accurately measure power consumption and relate that consumption to a particular virtual machine or application, but the power consumption of a given application or service may be heavily dependent on the demand and input profile for that service. Given that there are a large number of potential execution scenarios, some form of sampling is inevitable; this raises the issue of how best to choose that sample. Too large a sample may lead to an over-expensive evaluation function that cannot be practically accommodated within the constraints imposed by many SBSE algorithms. Search algorithms typically require large numbers of evaluations, so evaluation has to be efficient. However, too small a sample may yield an estimate of quality that may provide insufficient guidance for the search.

Finding the right sample size will, itself, be a matter of tuning pragmatics or theoretical analysis for the SBSE approach used, but this is not insurmountable. Compared to previous SBSE work, the process of designing an efficient evaluation function will require more effort.

The variance in performance characteristics over the sample also raises the issue of what form of aggregation is most appropriate. This will depend on the applications in hand and will be influenced by the target properties of the service level agreement between cloud provider and client. For instance, on some cloud settings, it will be important to reduce the variance in performance to increase predictability of execution. However, in other cases, the engineer may seek to reduce the average load or the peak load on resources such as time and memory.

6.3. Online optimisation

Many applications of SBSE to cloud systems will require *online* optimisation rather than offline optimisation, by performing *in situ* adaptation in the live environment. This is in contrast to most previous work, which has assumed pre-deployment optimisation in a clean-room environment. Online optimisation is yet to be seriously adopted by the SBSE community, and a major inhibiting factor is the cost of computation: optimisation methods such as evolutionary algorithms can require significant computational resources themselves.

A possible solution to this problem has recently been proposed, in the form of Amortised Optimisation (Yoo, 2012). Yoo suggests distributing the cost of optimisation across multiple executions of the target system: each execution of the target system provides a single observation of the evaluation function. While the optimisation process takes place over longer timescales, the total amount of computational overhead for the optimisation is significantly reduced, allowing it to take place in the live environment.

Cloud systems provide an amenable environment for amortised optimisation, as the parallel execution of the same application across multiple virtual machines presents the possibility of speeding up the amortised optimisation by aggregating many parallel observations of the evaluation function. This is analogous to past use of island models in distribution optimisation algorithms (Whitley, 2001).

7. Conclusions

Cloud computing offers the SBSE research agenda a stimulus of new challenges and opportunities. There is the challenge of using SBSE to address the trade-offs inherent in cloud computing. There are also new opportunities for extending past work in testing and maintenance within SBSE to cloud-specific applications.

Though the fundamental technical concepts of the cloud paradigm will be familiar to many, the emphasis they place on the combination of virtualisation, dynamic re-assignment, distribution and parallelism does raise important new research questions. This novel shift of emphasis also lends additional importance to many already known, but underdeveloped, applications of software engineering optimisation.

The shift to a pay-as-you-go business model for the deployment and use of computation also creates new challenges and opportunities. This shift is a 'disruptive innovation', because the cost of computation can more readily be measured directly in monetary terms. This is a significant change in the underlying computing business model. One might imagine that real-currency costing may ultimately have an impact on global finance. We may soon see trading in 'computational futures' and, perhaps, even the cost of computation as a 'global currency standard'.

The ability to directly measure cost will also have a profound effect on research, and particularly on SBSE research. Hitherto, cost measurement has proved to be a thorny problem in software engineering research: our forebears were forced to rely on cost surrogates. For example, those who sought to measure development cost had to be satisfied with function points, code size and person months of developer effort. Similarly, in place of test cost, we have become accustomed to using the test suite size, execution time and test process duration. Cloud computing may change this: development and test cost can be measured in dollars or yuan.

Most incarnations of SBSE are manifestations of the trade-off between cost and value objectives. For these SBSE formulations, the ability to measure cost in terms of the direct bottom line for the organisation is likely to add realism and actionability. Results for optimisation that produce reduced costs can be directly measured in financial gain to the adopters, making the business case for adoption simpler and more compelling.

The outputs of SBSE research for cloud are also likely to be of high impact because of the wide uptake of cloud research. Evidence of the growth of interest and activity in cloud computing abound. In this paper we have also presented evidence that the problems of cloud engineering are highly amenable to SBSE solutions, reformulating five such problems as SBSE problems. The rapid uptake of cloud technologies will mean that research advances will be likely

to impact on practice. However, the generic nature of many of these challenges, such as the virtualisation and distribution of computation, will also ensure that research will be highly transferable.

Despite enthusiasm for research on SBSE for the cloud, a question may remain in the mind of the would-be researcher: 'surely I need to own a cloud in order to do research in SBSE for the cloud?'. Fortunately, the nature of many of the cloud challenges means that they can be developed and evaluated without the need for a cloud system. For example, for specialised virtualisation, evaluation can be conducted on the degree to which the specialisation meets the needs of use case. Such evaluation can compare desktop versions (with and without specialisation) and need not, necessarily, be executed in the cloud to provide meaningful results.

In conclusion, there is a compelling case for SBSE for the cloud. The amenability of SBSE, the directness of the relationship to the business bottom line and likely impact are all motivations for the research agenda advocated in this paper. The barriers to researchers in terms of experimentation and evaluation are also not as great as might be presumed; cloud ownership is not a pre-requisite for cloud research.

Cloud computing is all about a multi-objective balance between conflicting and competing objectives. These are exactly the kinds of problems that have made SBSE attractive to software engineers. Multi-objective trade-offs also succinctly capture what it is that makes software engineering *engineering*; engineering is all about *optimisation*.

Acknowledgements

We gratefully acknowledge the invaluable assistance of our colleagues who read and commented on earlier drafts of this paper: Robert Feldt, Dimitrios Pezaros, Tim Storer, Richard Torkar, Posco Tso, and Westley Weimar. This research was partly supported by grants from the Engineering and Physical Sciences Research Council, and by the Scottish Informatics and Computer Science Alliance (SICSA).

References

Abadi, D.J., 2009. Data management in the cloud: limitations and opportunities. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 32, 3–12.

Afzal, W., Torkar, R., 2011. On the application of genetic programming for software engineering predictive modeling: a systematic review. Expert Systems Applications 38, 11984–11997.

Afzal, W., Torkar, R., Feldt, R., 2009. A systematic review of search-based testing for non-functional system properties. Information and Software Technology 51, 957–976.

Afzal, W., Torkar, R., Feldt, R., Wikstrand, G., 2010. Search-based prediction of fault-slip-through in large software projects. In: Second International Symposium on Search Based Software Engineering (SSBSE 2010).

Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K., 2010. A systematic review of the application and empirical investigation of search-based test-case generation. IEEE Transactions on Software Engineering 36, 742–762.

Alshahwan, N., Harman, M., 2011. Automated web application testing using search based software engineering. In: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011).

Amazon, 2012. Amazon EC2 Spot Instances. http://aws.amazon.com/ec2/spot-instances/ (accessed on 12.06.12).

Arcuri, A., Yao, X., 2008. A novel co-evolutionary approach to automatic software bug fixing. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08).

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2010. A view of cloud computing. Communications of the ACM 53, 50–58.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Zaharia, M., 2009. Above the clouds: a Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28. EECS Department, University of California, Berkeley.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization. SIGOPS Operating Systems Review 37, 164–177.

Barroso, L.A., Hölzle, U., 2009. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis Lectures on Computer Architecture 4.1, 1–108.

Beckman, L., Haraldson, A., Oskarsson, O., Sandewall, E., 1976. A partial evaluator, and its use as a programming tool. Artificial Intelligence 7, 319–357.

Beloglazov, A., Buyya, R., 2010. Energy efficient allocation of virtual machines in cloud data centers. In: IEEE/ACM International Conference on Cluster Cloud and Grid Computing.

Ben-Yehuda, O.A., Ben-Yehuda, M., Schuster, A., Tsafrir, D., 2011. Deconstructing Amazon EC2 spot instance pricing. In: IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom 2011).

Bjorner, D., Jones, N.D., Ershov, A.P. (Eds.), 1988. Partial Evaluation and Mixed Computation: Proceedings of the IFIP TC2 Workshop, Gammel Avernaes, Denmark, 18–24 Oct., 1987. Elsevier Science Inc., New York, NY, USA.

Cadar, C., Godefroid, P., Khurshid, S., Păsăreanu, C.S., Sen, K., Tillmann, N., Visser, W., 2011. Symbolic execution for software testing in practice: preliminary assessment. In: 33rd International Conference on Software Engineering (ICSE'11).

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software-Practice & Experience 41, 23–50.

Carzaniga, A., Gorla, A., Pezzé, M., 2008. Healing web applications through automatic workarounds. International Journal on Software Tools for Technology Transfer 10, 493–502.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E., 2008. Bigtable: a distributed storage system for structured data. ACM Transactions on Computer Systems 4 (1–4), 26.

Cheng, B., Atlee, J., 2007. From state of the art to the future of requirements engineering. In: Future of Software Engineering 2007.

Cliff, D., 2010. Remotely hosted services and 'Cloud Computing'. British Educational Communications and Technology Agency (BECTA).

Cohen, K.J., Maier, S.F., Schwartz, R.A., Whitcomb, D.K., 1983. A simulation model of stock exchange trading. Simulation 41, 181–191.

Cooper, K.D., Schielke, P.J., Subramanian, D., 1999. Optimizing for reduced code space using genetic algorithms. In: Proceedings of the ACM Sigplan 1999 Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'99).

Cornford, S.L., Feather, M.S., Dunphy, J.R., Salcedo, J., Menzies, T., 2003. Optimizing spacecraft design – optimization engine development: progress and plans. In: Proceedings of the IEEE Aerospace Conference.

Darley, V., Outkin, A.V., 2007. Nasdaq Market Simulation: Insights on a Major Market from the Science of Complex Adaptive Systems. World Scientific Publishing Company.

De Millo, R.A., Lipton, R.J., Perlis, A.J., 1979. Social processes and proofs of theorems and programs. Communications of the ACM 22, 271–280.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W., 2007. Dynamo: Amazon's highly available key-value store. In: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles.

Dijkstra, E.W., 1978. On a political pamphlet from the middle ages (a response to the paper 'Social Processes and Proofs of Theorems and Programs' by DeMillo, Lipton, and Perlis). ACM SIGSOFT, Software Engineering Notes 3, 14–17.

Durillo, J.J., Zhang, Y., Alba, E., Nebro, A.J., 2009. A study of the multi-objective next release problem. In: Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09).

Emberson, P., Bate, I., 2010. Stressing search with scenarios for flexible solutions to real-time task allocation problems. IEEE Transactions on Software Engineering 36, 704–718.

Fan, X., Weber, W., Barroso, L.A., 2007. Power provisioning for a warehouse-sized computer. SIGARCH Computer Architecture News 35, 13–23.

Fatiregun, D., Harman, M., Hierons, R., 2005. Search-based amorphous slicing. In: 12th International Working Conference on Reverse Engineering (WCRE 05).

Forrest, S., Nguyen, T., Weimer, W., Le Goues, C., 2009. A genetic programming approach to automated software repair. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 947–954.

Fraser, G., Arcuri, A., 2011. EvoSuite: automatic test suite generation for object-oriented software. In: 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11).

Freitas, F.G., Souza, J.T., 2011. Ten years of search based software engineering: a bibliometric analysis. In: 3rd International Symposium on Search based Software Engineering (SSBSE 2011).

Google, Inc., 2012. Traces of Google Workloads. http://code.google.com/p/googleclusterdata/ (accessed on 12.06.12).

Guo, P., Engler, D., 2011. CDE: using system call interposition to automatically create portable software packages. In: Proceedings of the USENIX Annual Technical Conference, pp. 247–252.

Harman, M., 2007. Automated test data generation using search based software engineering. In: 2nd International Workshop on Automation of Software Test (AST 07).

Harman, M., 2007. Search based software engineering for program comprehension. In: 15th International Conference on Program Comprehension (ICPC 07).

Harman, M., 2007. The current state and future of search based software engineering. In: Future of Software Engineering 2007.

Harman, M., 2008. Open problems in testability transformation. In: 1st International Workshop on Search Based Testing (SBT 2008).

Harman, M., 2010. The relationship between search based software engineering and predictive modeling. In: 6th International Conference on Predictive Models in Software Engineering.
Harman, M., 2010. Why source code analysis and manipulation will always be important. In: 10th IEEE International Working Conference on Source Code Analysis and Manipulation.
Harman, M., Hierons, R.M., 2001. An overview of program slicing. Software Focus 2, 85–92.
Harman, M., Jones, B.F., 2001. Search based software engineering. Information and Software Technology 43, 833–839.
Harman, M., Mansouri, S.A., Zhang, Y., 2012a. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys 45, 11:1–11:61.
Harman, M., McMinn, P., 2010. A theoretical and empirical study of search based testing: local, global and hybrid search. IEEE Transactions on Software Engineering 36, 226–247.
Harman, M., McMinn, P., Souza, J., Yoo, S., 2012b. Search based software engineering: techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification: LASER 2009–2010, pp. 1–59.
Hoare, C.A.R., 1978. The engineering of software: a startling contradiction. In: Programming Methodology, A Collection of Articles by Members of IFIP WG2.3.
Hoare, C.A.R., 1996. How did software get so reliable without proof? In: IEEE International Conference on Software Engineering (ICSE'96).
Hoare, C.A.R., 1996. How did software get so reliable without proof? In: FME '96: Industrial Benefit and Advances in Formal Methods: Third International Symposium of Formal Methods Europe.
Hoste, K., Eeckhout, L., 2008. Cole: compiler optimization level exploration. In: Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization.
IDC, 2011. Press Release. http://www.idc.com/getdoc.jsp?containerId=prUS23177411 (accessed on 12.06.12).
Jacobs, A., 2009. The pathologies of big data. Communications of the ACM 52, 36–44.
Jakobović, D., Budin, L., 2006. Dynamic scheduling with genetic programming. In: EuroGP 2006.
Jakobovic, D., Jelenkovic, L., Budin, L., 2007. Genetic programming heuristics for multiple machine scheduling. In: EuroGP 2007.
Jia, Y., Harman, M., 2008. Milu: a customizable, runtime-optimized higher order mutation testing tool for the full C language. In: 3rd Testing Academia and Industry Conference – Practice and Research Techniques (TAIC PART'08).
Jones, N.D., 1996. An introduction to partial evaluation. ACM Computing Surveys 28, 480–503.
Justafort, V.D., 2012. Performance-aware virtual machine allocation approach in an intercloud environment. In: Electrical & Computer Engineering (CCECE) 2012.
Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220, 671–680.
Kliazovich, D., Bouvry, P., Audzevich, Y., Khan, S., 2010. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. In: Global Telecommunications Conference (GLOBECOM 2010), IEEE.
Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge.
Lagar-cavilla, H.A., Whitney, J.A., Scannell, A., Patchin, P., Rumble, S.M., Lara, E.D., Brudno, M., Satyanarayanan, M., 2009. SnowFlock: rapid virtual machine cloning for cloud computing. In: Proceeding of the EuroSys.
Lakhotia, K., Harman, M., Gross, H., 2010a. AUSTIN: a tool for search based software testing for the c language and its evaluation on deployed automotive systems. In: 2nd International Symposium on Search Based Software Engineering (SSBSE 2010).
Lakhotia, K., Tillmann, N., Harman, M., de Halleux, J., 2010b. FloPSy – search-based floating point constraint solving for symbolic execution. In: 22nd IFIP International Conference on Testing Software and Systems (ICTSS 2010).
Langdon, W.B., Harman, M., 2010. Evolving a CUDA kernel from an nVidia template. In: IEEE Congress on Evolutionary Computation.
Le Goues, C., Dewey-Vogt, M., Forrest, S., Weimer, W., 2012. A systematic study of automated program repair: fixing 55 out of 105 bugs for $8 each. In: International Conference on Software Engineering.
Le Goues, C., Nguyen, T., Forrest, S., Weimer, W., 2012. GenProg: a generic method for automatic software repair. IEEE Transactions on Software Engineering 38, 54–72.
Lee, Y.C., Zomaya, A.Y., 2012. Energy efficient utilization of resources in cloud computing systems. Journal of Supercomputing 60, 268–280.
Lutz, R., 2001. Evolving good hierarchical decompositions of complex systems. Journal of Systems Architecture 47, 613–634.
Madhavapeddy, A., Mortier, R., Sohan, R., Gazagnaire, T., Hand, S., Deegan, T., McAuley, D., Crowcroft, J., 2010. Turning down the LAMP: software specialisation for the cloud. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.
McMinn, P., 2004. Search-based software test data generation: a survey. software testing. Verification and Reliability 14, 105–156.
Mell, P., Grance, T., 2009. The NIST definition of cloud computing. National Institute of Standards and Technology 53, NIST Special Publication 800-145.
Microsoft, 2012. Autoscaling Rules Schema Description. http://msdn.microsoft.com/en-us/library/hh680955 (accessed on 12.06.12).
Mishra, A.K., Hellerstein, J.L., Cirne, W., Das, C.R., 2010. Towards characterizing cloud backend workloads: insights from Google compute clusters. SIGMETRICS Performance Evaluation Review 37, 34–41.
Mitchell, B.S., Mancoridis, S., 2006. On the automatic modularization of software systems using the bunch tool. IEEE Transactions on Software Engineering 32, 193–208.
Narzisi, G., Mysore, V., Mishra, B., 2006. Multi-objective evolutionary optimization of agent-based models: an application to emergency response planning. Computational Intelligence, 224–230.
Netflix Tech Blog, 2008. 5 Lessons We've Learned Using AWS. http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html (accessed on 12.06.12).
Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D., 2009. The eucalyptus open-source cloud-computing system. In: Cluster Computing and the Grid.
Panayi, E., Harman, M., Wetherilt, A. Agent-based modelling of stock markets using existing order book data. In: Proceedings of 13th International Workshop on Multi-Agent-Based Simulation (MABS), in press.
Papazoglou, M., van den Heuvel, W.J., 2007. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16, 389–415.
Räihä, O., 2010. A survey on search-based software design. Computer Science Review 4, 203–249.
Rappa, M., 2004. The utility business model and the future of computing services. IBM Systems Journal 43, 32–42.
Red Hat Enterprise, 2007. Oracle Tuning Guide. http://docs.redhat.com/docs/en-US/Red_hat_Enterprise_Linux/5/html/Oracle_Tuning_Guide/RHELTuningandOptimizationforOracleV11.pdf (accessed on 12.06.12).
Reese, G., 2009. Cloud Application Architectures. O'Reilly.
Rogers, A., von Tessin, P., 2004. Multi-objective calibration for agent-based models. In: 5th Workshop on Agent-based Simulation.
Ryan, C., 2000. Automatic Re-engineering of Software Using Genetic Programming. Kluwer Academic Publishers.
Silva, J., 2012. A vocabulary of program slicing-based techniques. ACM Computing Surveys 44, 12:1–12:41.
Sitthi-Amorn, P., Modly, N., Weimer, W., Lawrence, J., 2011. Genetic programming for shader simplification. ACM Transactions on Graphics 30, 152:1–152:12.
Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I., 2009. Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing 13, 14–22.
Srikantaiah, S., Kansal, A., Zhao, F., 2008. Energy aware consolidation for cloud computing. In: HotPower'08.
Stillwell, M., Vivien, F., Casanova, H., 2012. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In: Parallel Distributed Processing Symposium (IPDPS) 2012.
Viegas, F., Wattenberg, M., Feinberg, J., 2009. Participatory visualization with wordle. IEEE Transactions on Visualization and Computer Graphics 15, 1137–1144.
Vishwanath, K.V., Nagappan, N., 2010. Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM symposium on Cloud Computing.
Voorsluys, W., Broberg, J., Venugopal, S., Buyya, R., 2009. Cost of virtual machine live migration in clouds: a performance evaluation. In: Proceedings of the 1st International Conference on Cloud Computing.
Wegener, J., Bühler, O., 2004. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In: Genetic and Evolutionary Computation Conference (GECCO 2004).
Weimer, W., Nguyen, T.V., Le Goues, C., Forrest, S., 2009. Automatically finding patches using genetic programming. In: International Conference on Software Engineering (ICSE 2009).
Weiser, M., 1979. Program slices: formal, psychological, and practical investigations of an automatic program abstraction method. Ph.D. Thesis. University of Michigan, Ann Arbor, MI.
White, D., Arcuri, A., Clark, J., 2011. Evolutionary improvement of programs. IEEE Transactions on Evolutionary Computation 15, 515–538.
White, D.R., Clark, J., Jacob, J., Poulding, S., 2008. Searching for resource-efficient programs: low-power pseudorandom number generators. In: 2008 Genetic and Evolutionary Computation Conference (GECCO 2008).
Whitley, D., 2001. An overview of evolutionary algorithms: practical issues and common pitfalls. Information and Software Technology 43, 817–831.
Williams, D., Jamjoom, H., Liu, Y., Weatherspoon, H., 2011. Overdriver: handling memory overload in an oversubscribed cloud. SIGPLAN Notices 46, 205–216.
Wine, 2012. Windows Compatibility Layer for UNIX. http://www.winehq.org/ (accessed on 12.06.12).
Yoo, S., 2012. NIA^{3}CIN: non-invasive autonomous and amortised adaptivity code injection. Technical Report RN/12/13. Department of Computer Science, University College London.
Yoo, S., Nilsson, R., Harman, M., 2011. Faster fault finding at Google using multi-objective regression test optimisation. In: 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11).
Zhang, Y., Finkelstein, A., Harman, M., 2008. Search based requirements optimisation: existing work and challenges. In: International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'08).
Zhang, Y., Harman, M., Mansouri, A., 2012. The SBSE Repository: A Repository and Analysis of Authors and Research Articles on Search Based Software Engineering. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/
Zhang, Y., Harman, M., Mansouri, S.A., 2007. The multi-objective next release problem. In: GECCO '07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference.

Mark Harman is professor of Software Engineering in the Department of Computer Science at University College London, where he directs the CREST centre and is Head of Software Systems Engineering. He is widely known for work on source code analysis and testing and was instrumental in founding the field of Search Based Software Engineering (SBSE), the topic of this paper. SBSE research has rapidly grown over the past five years and now includes over 800 authors, from nearly 300 institutions spread over more than 40 countries. A recent tutorial paper on SBSE can be found here: http://www.cs.ucl.ac.uk/staff/mharman/laser.pdf.

Kiran Lakhotia is a Research Associate in the CREST centre at University College London. He is working in the field of Search-Based Software Testing and in particular automated test data generation. In 2009 he received his PhD in Computer Science from King's College London.

Jeremy Singer is a lecturer in Complex Systems Engineering at the University of Glasgow. He received Bachelor and PhD degrees in Computer Science from the University of Cambridge. His research interests include optimizing compilation, virtual machines, memory management and many core parallelism.

David R White is a SICSA Research Fellow in Complex Systems within the School of Computing at the University of Glasgow. His work includes the creation and optimisation of software using heuristic search and evolutionary computing, and new ways of researching and teaching cloud computing. He received the PhD degree in Computer Science from the University of York in 2010 for his work on applying search based software engineering methods to the optimisation of software properties such as power consumption and execution time.

Shin Yoo is a lecturer of software engineering in the Centre for Research in Evolution, Search, and Testing at University College London. He received his PhD in Computer Science from King's College London in 2009. He is working on the application of meta-heuristic optimisation and information theory for software testing.